FIG. 1

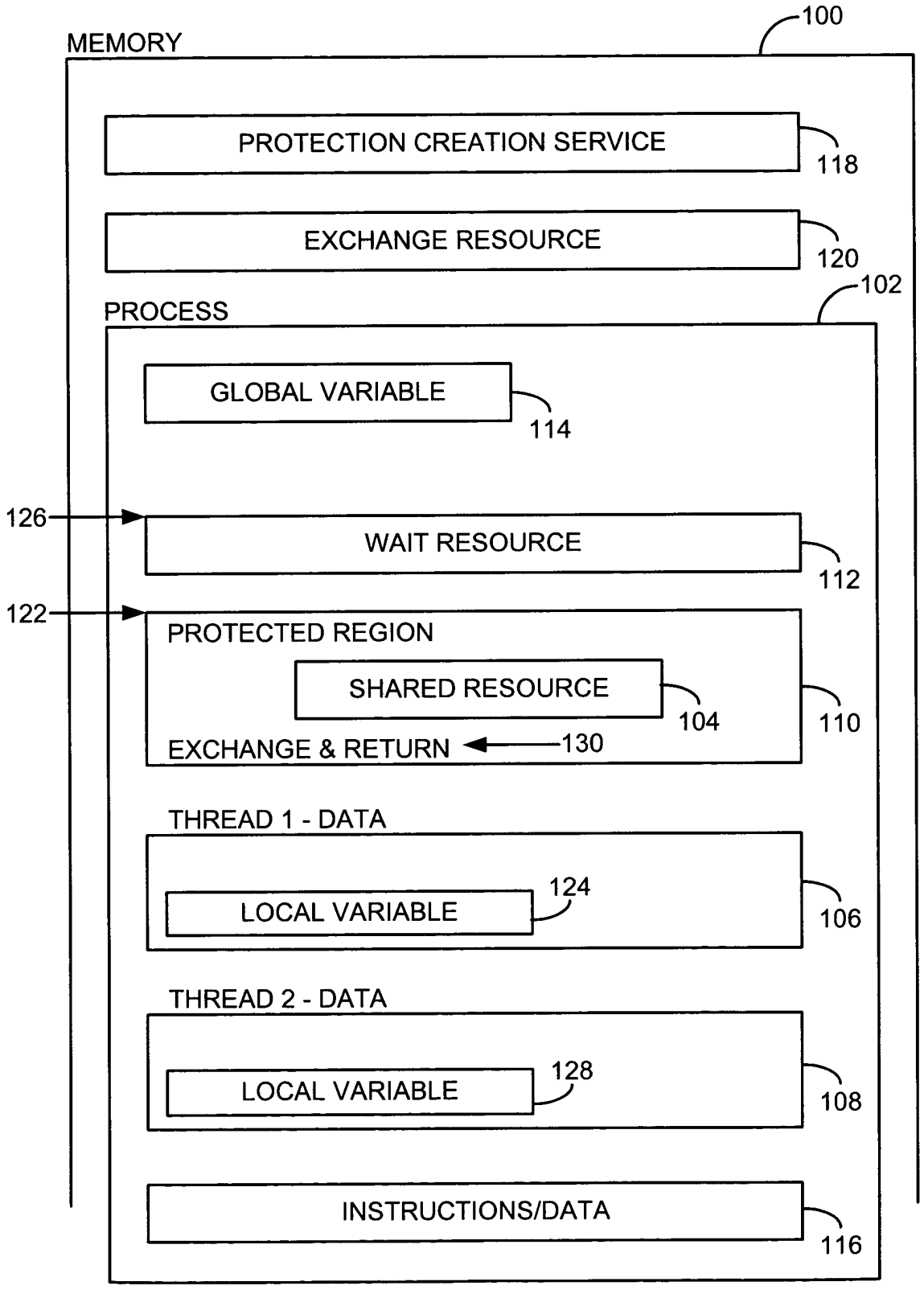

MEMORY
100
PROTECTION CREATION SERVICE
118
EXCHANGE RESOURCE
120
PROCESS
102
GLOBAL VARIABLE
114
126
WAIT RESOURCE
112
122
PROTECTED REGION
SHARED RESOURCE
104
110
EXCHANGE & RETURN
130
THREAD 1 - DATA
LOCAL VARIABLE
124
106
THREAD 2 - DATA
LOCAL VARIABLE
128
108
INSTRUCTIONS/DATA
116

FIG. 2

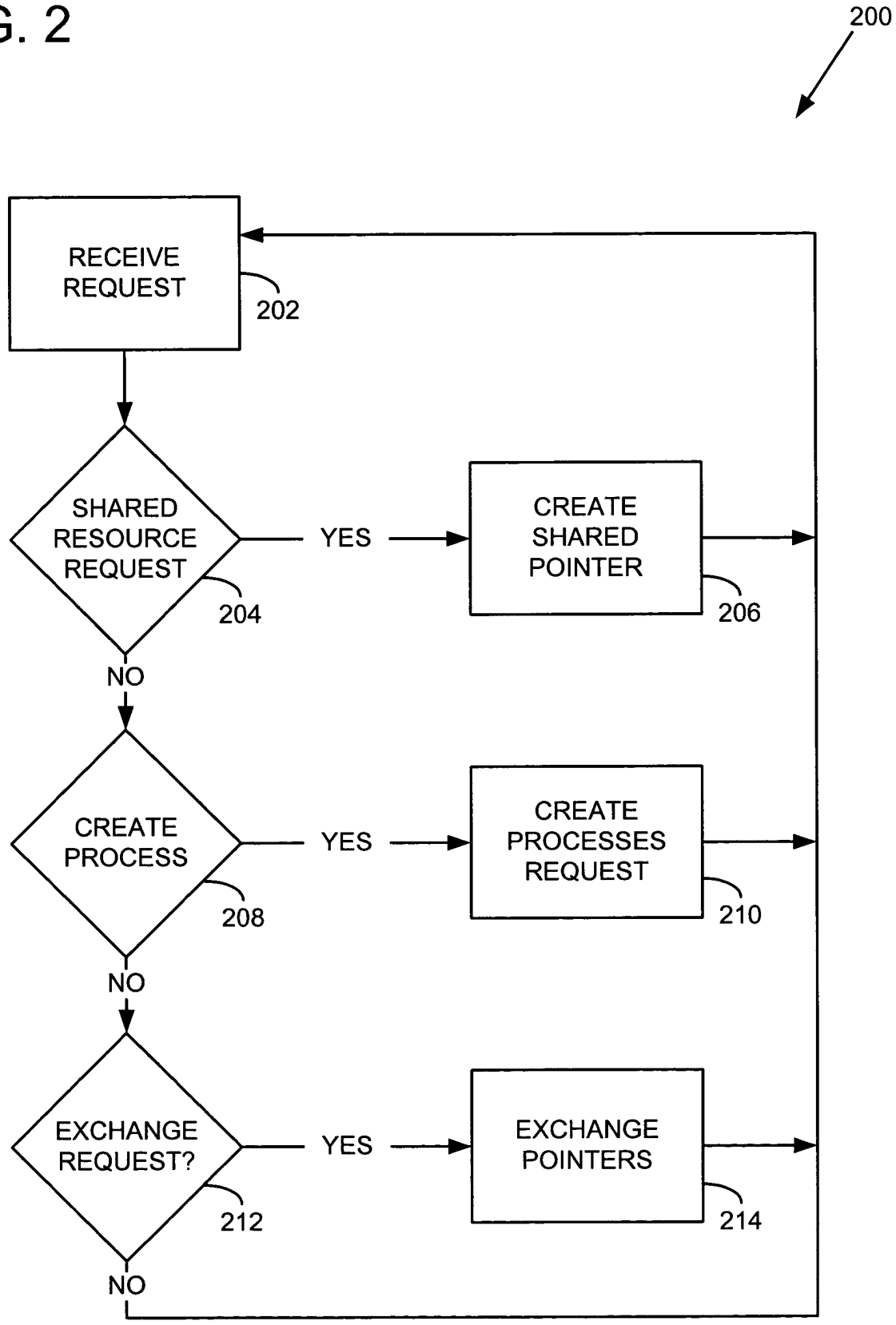

200
RECEIVE REQUEST
202
SHARED RESOURCE REQUEST
204
YES
CREATE SHARED POINTER
206
NO
CREATE PROCESS
208
YES
CREATE PROCESSES REQUEST
210
NO
EXCHANGE REQUEST?
212
YES
EXCHANGE POINTERS
214
NO

FIG. 3

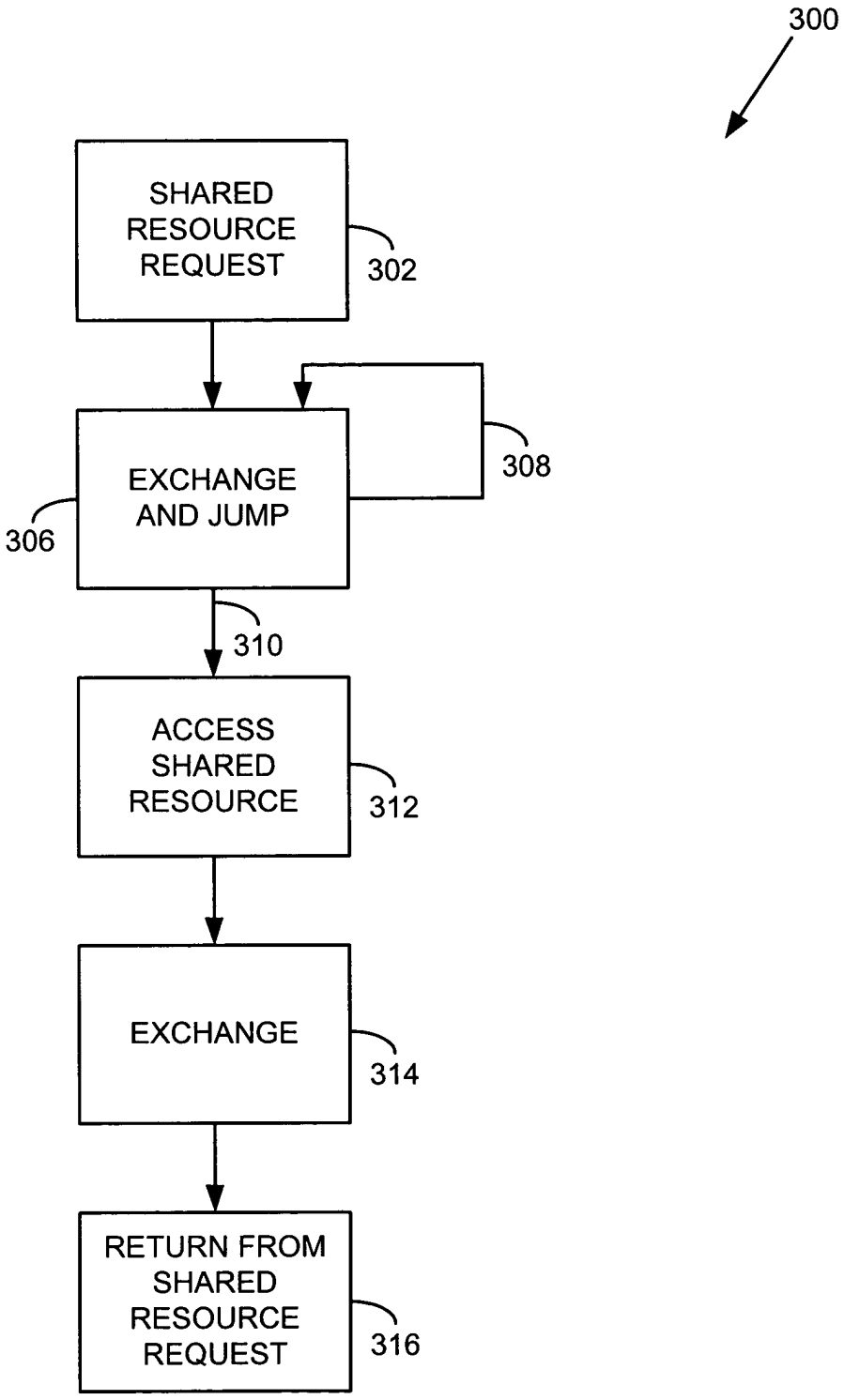

300
SHARED RESOURCE REQUEST
302
EXCHANGE AND JUMP
306
308
310
ACCESS SHARED RESOURCE
312
EXCHANGE
314
RETURN FROM SHARED RESOURCE REQUEST
316

FIG. 4

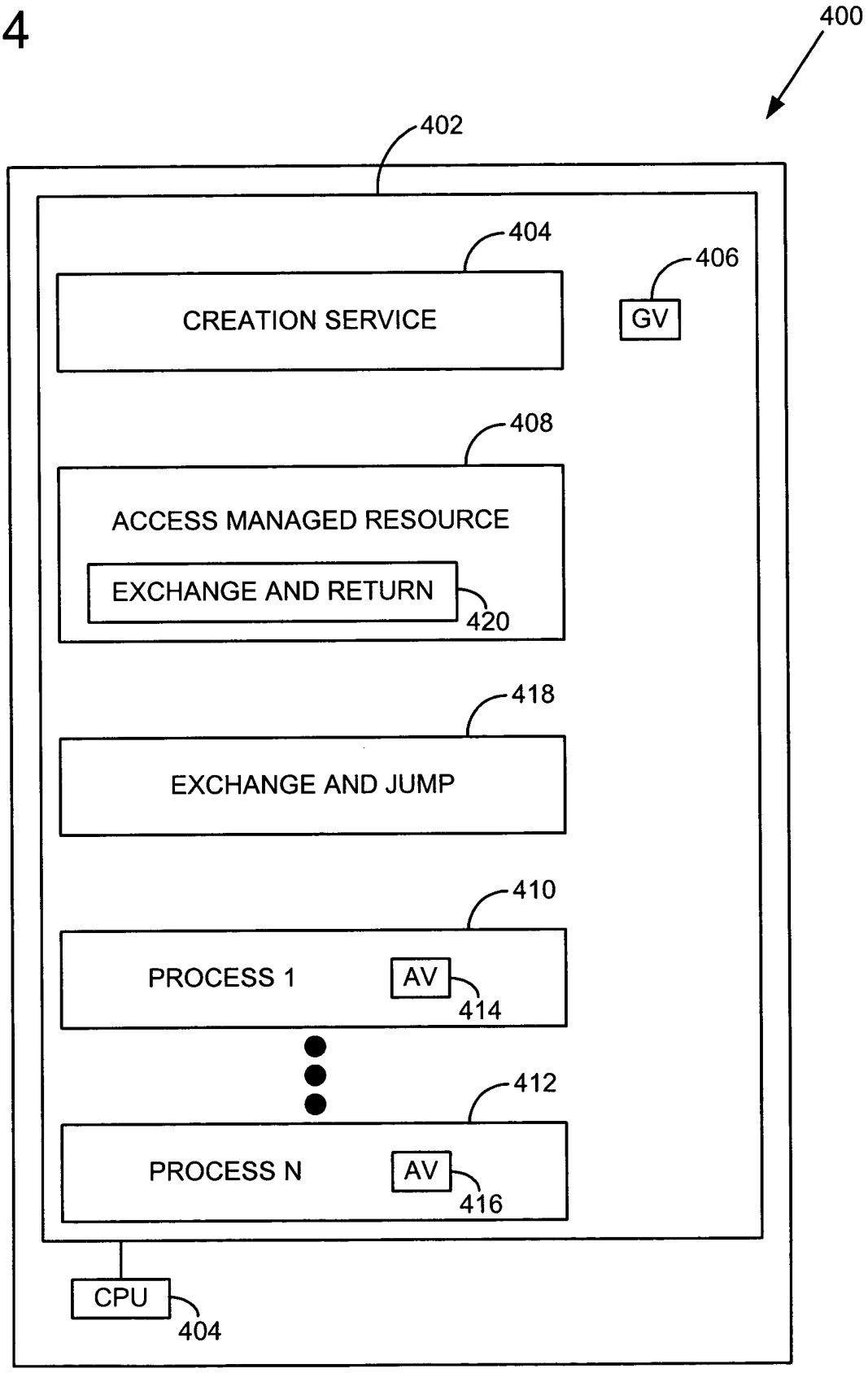

400
402
404
CREATION SERVICE
406
GV
408
ACCESS MANAGED RESOURCE
EXCHANGE AND RETURN
420
418
EXCHANGE AND JUMP
410
PROCESS 1
AV
414
412
PROCESS N
AV
416
CPU
404

FIG. 5

500

lock_sample.cpp ← 502

```
#include "lock.h" ← 504

typedef class CAquireLock<CSpinLock> CAquireSpinLock; ← 506

// This is the lock variable, which can be aquired.
CSpinLock m_TheLock; ← 508

// This is a globally shared data variable that should be used carefully to
// avoid duplicate IDs.  See code below.
long GlobalTransactionID = 0; ← 510

void ProcessBankTransaction ← 512
(
  BankTransaction * pTransaction
)
{
  // First aquire the lock so I can process the bank transaction
  // without conflicting with other bank transactions

  CAquireSpinLock lock(&m_TheLock); ← 514

  // Lock is now aquired, I can now process the transaction safely

  // Assign transaction a unique transaction ID
  // This cant be done without a lock. Without a lock, its possible that
  // 2 different transactions could have the same transaction ID.
  GlobalTransactionID = GlobalTransactionID + 1; ← 516
  pTransaction->ID = GlobalTransactionID;

  // Now do more transaction processing like a withdrawl, deposit, transfer, ...
  // ... code omitted for simplicity


  // Done with the code
  // Compiler will insert a call to the CAquireSpinLock destructor, which will
  // unlock the aquired lock (m_TheLock)
  return; ← 518
}
```

FIG. 6

600

lock.h

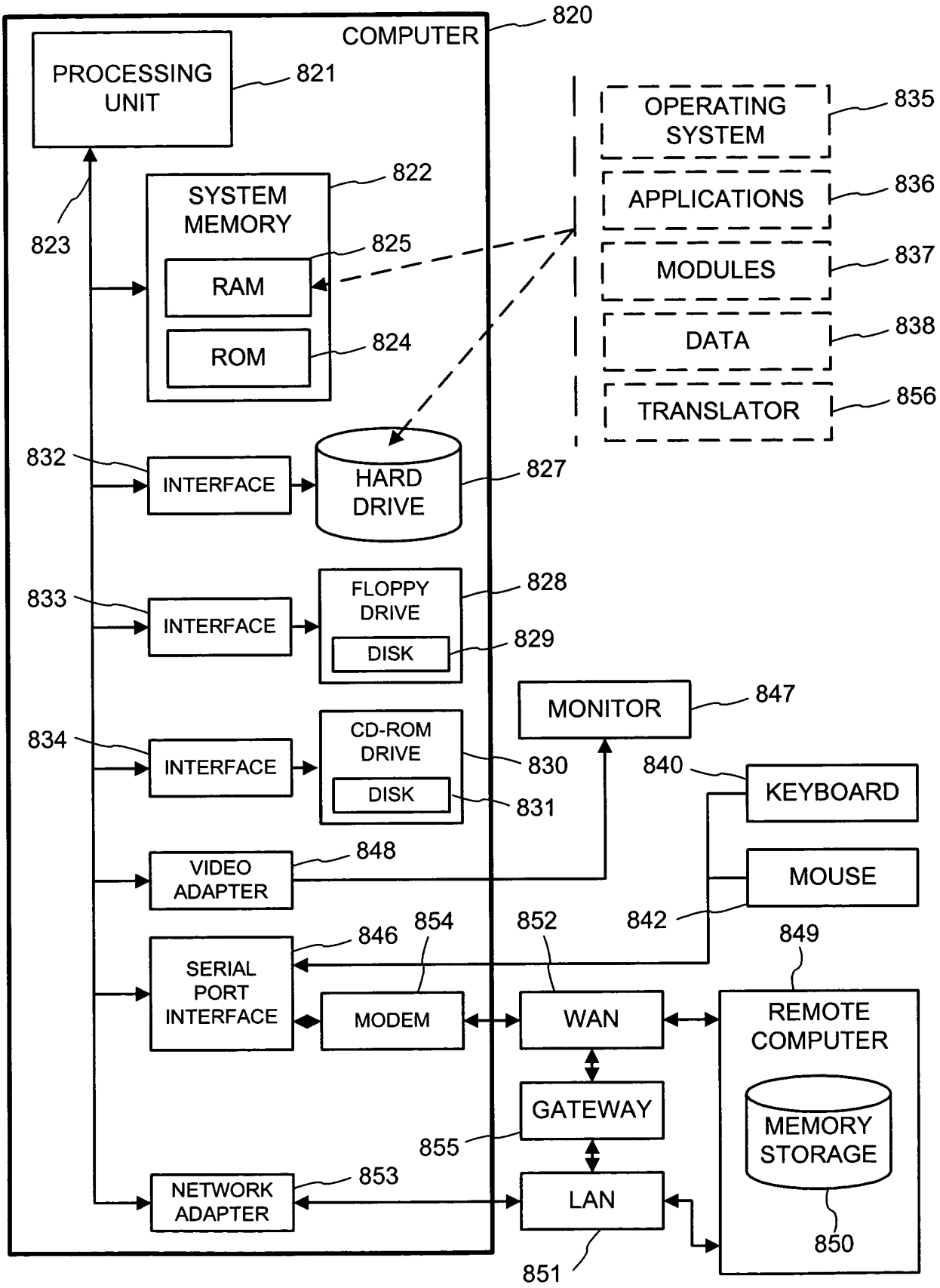
```
void __declspec(naked) WaitForLock() <-- 614
{
  __asm
  {
      // only 2 destinations: Spin or Aquired.  Aquired is the only way
      // to get out of here.  Spin just keeps looping until lock is aquired.

      push      ebx <-- 616
      mov       ebx,  [esp + 8] <-- 618

  Spin: <-- 620
      mov       eax, Spin <-- 622
      lock xchg   eax,  dword ptr [ebx] <-- 624
      jmp       eax <-- 626
  }
}
inline void __declspec(naked) Aquired() <-- 628
{
  __asm
  {
    pop     ebx <-- 630
    ret     4 <-- 632
  }
}
class CSpinLock
{
public:
  CSpinLock() <-- 602
  {
    // Initial lock variable to Aquired function.  When the Aquired function
    // is executed, then this class has aquired the lock.
    m_pv = Aquired; <-- 604
  }
  void Lock() <-- 606
  {
    void ** ppv = &m_pv; <-- 608

    __asm
    {                          <-- 610
      push  ppv          // pass pointer to lock variable to WaitForLock.
      call  WaitForLock    // call function to wait until lock is aquired
    }                          <-- 612
  }
```

FIG. 7

700

```
  void Unlock() <---- 718
  {
    void ** ppv = &m_pv;
    __asm
    {
        push  ebx <---- 720
        mov   ebx,    ppv <---- 722
        mov   eax,    Aquired <---- 724
    lock xchg eax,    [ebx] <---- 726
        pop   ebx <---- 728
    }
  }
protected:
  void * m_pv; <---- 730
};
template <class CLockWorker>
class CAquireLock
{
public:
  CAquireLock(CLockWorker * plock) <---- 702
  {
    m_plock = plock; <---- 704
    Lock(); <---- 706
  }
  ~CAquireLock() <---- 710
  {
    Unlock(); <---- 712
    m_plock = NULL;
  }
  void Lock()
  {
    m_plock->Lock(); <---- 708
  }
  void Unlock() <---- 714
  {
    m_plock->Unlock(); <---- 716
  }
protected:
  CLockWorker * m_plock;
};
```

FIG. 8


COMPUTER
820
PROCESSING UNIT
821
SYSTEM MEMORY
822
823
RAM
825
ROM
824
OPERATING SYSTEM
835
APPLICATIONS
836
MODULES
837
DATA
838
TRANSLATOR
856
832
INTERFACE
HARD DRIVE
827
833
INTERFACE
FLOPPY DRIVE
828
DISK
829
834
INTERFACE
CD-ROM DRIVE
830
DISK
831
MONITOR
847
840
KEYBOARD
MOUSE
842
VIDEO ADAPTER
848
SERIAL PORT INTERFACE
846
854
852
849
MODEM
WAN
REMOTE COMPUTER
GATEWAY
MEMORY STORAGE
855
NETWORK ADAPTER
853
LAN
850
851